Vol. 35 No. 8　　情　報　処　理　　Aug. 1994

解　説

論理設計の形式的検証

1. 論理関数処理に基づく形式的検証手法†

平石裕実††　浜口清治†††

1. まえがき

設計される論理回路の大規模化や多様化にともない，設計された回路が，与えられた仕様を満足するかどうかを完全に保証するための形式的検証は，ますます重要になってきている．本稿では，特に同期式順序回路(以下，単に順序回路)を対象とした形式的検証のうち，時相論理を仕様記述言語とする検証手法と順序回路の等価性判定に基づく検証手法を取り上げ，近年，主流となりつつある，二分決定グラフの使用を前提とした論理関数処理に基づく手法について，その基本アルゴリズムといくつかの効率改善手法について解説する．

順序回路の等価性判定は，たとえば，状態遷移図やハードウェア記述言語で与えられた記述と，その実現としてのゲートレベルでの記述が等価であるかどうかを判定するために用いられ，双方の記述を状態遷移図の形式に変換して行われる．

一方，時相論理[1]は従来の命題論理に時間の概念を陽に表現するための時相演算子を付加した論理体系であるが，これを用いた順序回路（より一般には有限状態機械）の形式的設計検証では，仕様記述言語として時相論理の一種である計算木論理 (Computation Tree Logic, CTL) を用いたモデル検査手法によるものが，小規模な回路に対してではあるが，実用的に成功をおさめてきた[2),3)]．この手法は，検証対象である有限状態機械がCTLの論理式で記述された性質を満足するかどうかを調べるというものである．

これらの手法は，状態遷移図を構成することを前提としているが，この際問題となるのは，**状態数爆発 (state explosion) 問題**である．たとえば，フリップフロップ数 n の順序回路に対応する状態遷移図を構成すると，最悪の場合 2^n の状態数をもつことになる．このため，取り扱うことのできる設計の規模は，せいぜい 10^6 状態程度の回路にとどまっていた．

一方，Akers や Bryant らにより定式化された**二分決定グラフ (Binary Decision Diagram, BDD)**[4),5)]は，論理関数のグラフ表現であり，多くの実用的な論理関数を比較的コンパクトに表現できることや論理演算を高速に実行できるといった特徴をもち，論理設計 CAD における論理関数処理に広く用いられるようになってきている．

BDDを応用した形式的検証手法としては，Boseら[6)]やBurchら[7),8)]がCTLのモデル検査アルゴリズムを，またCoudertらが順序回路の等価性判定アルゴリズム[9),10)]を示し，いずれのアルゴリズムでも状態数が 10^{20} をこえるきわめて大きな順序回路の検証も可能になる場合があることを実証した．これらはいずれも状態集合や状態遷移関数を論理関数で表現し，アルゴリズムは全て論理関数操作により行うというものであり，本稿で紹介するのは，その基本アルゴリズムと効率改善手法である．

以下，**2.** でCTLや論理関数処理の諸定義を示した後，**3.** で論理関数処理による基本アルゴリズムを示し，**4.** で効率改善手法を紹介する．なお，本稿の内容に関連して全般的に良くまとめられた図書として文献11)も併せて参考にされたい．

2. 準　備

本稿では，二分決定グラフについての説明は行わないので，文献[5),12)~14)]を参照されたい．

† Formal Verification Methods Based on Logic Function Manipulation by Hiromi HIRAISHI (Department of Information & Communication Sciences, Faculty of Engineering, Kyoto Sangyo University) and Kiyoharu HAMAGUCHI (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都産業大学工学部情報通信工学科
††† 京都大学工学部情報工学科

2.1 Computation Tree Logic (CTL)

CTL は原始命題の真偽値の時間的な変化を記述できる時相論理体系であり，CTL 式の意味を定義するセマンティックモデルとしては，有限状態機械のモデルと考えられる Kripke 構造と呼ばれる有向グラフを用いる．

定義 1 (Kripke 構造) Kripke 構造は四つ組 $K=(V, R, I, V_0)$ で定義される．ここに，V は節点の有限集合，$R\subseteq V\times V$ は節点集合上の2項関係で，節点間の遷移関係を表す．任意の節点は少なくとも一つ以上の次節点（子節点）をもつものと仮定する．$I: V\to 2^{AP}$ は各節点で真になる原始命題の集合を与える．ただし，AP は原始命題の集合である．また，$V_0\subseteq V$ は初期節点の集合である．

CTL の論理演算子は，通常の命題論理の論理和(+)や論理積(・)，論理否定(¬)，含意(⇒)，等価（≡）などの命題演算子と時相演算子からなる．時相演算子は，Kripke 構造のある節点からの「全ての遷移系列に対して」という意味を表す**全称記号** (A) と「ある遷移系列に対して」という意味を表す**存在記号** (E) を用い，これらを時間に関する性質を表現する演算子(「次の時刻(next)」を意味する X,「将来いつか (future)」を意味する F,「将来常に (global)」を意味する G,「ある性質が成り立つまでは別の性質が成立し続ける(until)」を意味する U など）と組み合わせたものである．それらの直観的な意味は次のとおりである．

$EX\varphi$ ($AX\varphi$)：現在の節点のある（全ての）子節点で φ が成立する．

$EG\varphi$ ($AG\varphi$)：現在の節点からのある（全ての）遷移系列において，常に φ が成立する．

$EF\varphi$ ($AF\varphi$)：現在の節点からのある（全ての）遷移系列において，いつか φ が成立する．

$E[\eta U\varphi]$ ($A[\eta U\varphi]$)：現在の節点からのある（全ての）遷移系列において，いつか φ が成立し，かつ最初に φ が成立するまでは η が成立し続ける．

これらの時相演算子のうち，たとえば EX, EG, EU があれば，残りの時相演算子を表現することができる．なお，CTL のセマンティクスの形式的な定義については文献 2), 3) を参照されたい．

CTL を順序回路の仕様記述言語として用いる場合，回路の入力信号線や状態変数を原始命題として（たとえば0と1の2値をとる信号線 x は，1を「真」，0を「偽」と考えて CTL の原始命題とみなせる）仕様記述を行うことになる．

2.2 論理関数による表現と処理

$B=\{0,1\}$ をブール値の集合とする．このとき，$E=B^n$ の部分集合 A に対して，A の**特性関数 (characteristic function)** $\chi_A: E\to\{0,1\}$ は次の条件を満足する n 変数論理関数として定義される．

$$\chi_A(\vec{x})=1 \text{ iff } \vec{x}\in A$$

$f(\vec{x})$ を n 変数論理関数とする．このとき，**スムージング (smoothing) 演算子** $\exists x_i.$ や $\exists \vec{x}.$ は次のように定義される．

$$\exists x_i.f=f_{x_i}+f_{\overline{x_i}}$$

$$\exists \vec{x}.f=\exists x_1.\cdots\exists x_n.f$$

ここで，f_a と $f_{\bar{a}}$ は f の変数 a に1と0をそれぞれ代入して得られる論理関数を表す．

本稿では，**図-1** に示すような n 本の入力信号線，k 本の出力信号線，m 個のフリップフロップをもつ順序回路 $M=(\vec{x}, \vec{y}, \vec{f}(\vec{x}, \vec{y}), \vec{z}(\vec{x}, \vec{y}), \vec{f}_{init}(\vec{y}))$ の設計検証問題を扱う．ここに $\vec{x}=(x_1, \cdots, x_n)$ は入力変数ベクトル，$\vec{y}=(y_1, \cdots, y_m)$ は現状態変数ベクトル，$\vec{f}(\vec{x}, \vec{y})=(f_1(\vec{x}, \vec{y}), \cdots, f_m(\vec{x}, \vec{y}))$ は次状態関数ベクトル，$\vec{z}(\vec{x}, \vec{y})=(z_1(\vec{x}, \vec{y}), \cdots, z_k(\vec{x}, \vec{y}))$ は出力関数ベクトル，$f_{init}(\vec{y})$ は初期状態（の集合）

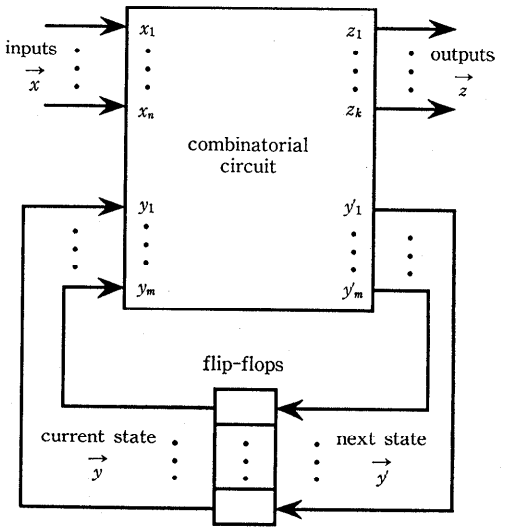


図-1 順序回路

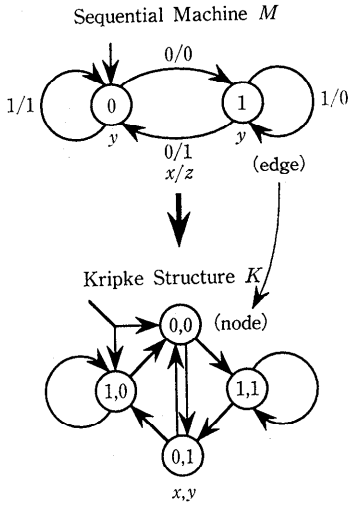


図-2　順序機械と Kripke 構造

を表す特性関数である．

$\vec{y}'$ を次状態変数ベクトル，$\vec{x}'$ を次時刻での入力変数ベクトルとする．また，$\vec{v}(\vec{v}')$ で $\vec{x}(\vec{x}')$ と $\vec{y}(\vec{y}')$ を連結したベクトルを表す．このとき，

$$R(\vec{v},\vec{v}')=\prod_{i=1,\cdots,m}(y_i'\equiv f_i(\vec{v}))$$

を**遷移関係関数 (transition relation function)** と呼ぶ．これは，**図-2** に示すように，順序回路の状態遷移図の枝（入力 $\vec{x}$ と状態 $\vec{y}$ の組）を節点 $(\vec{v})$ とみなして構成できる有向グラフの全ての枝の集合を表す特性関数とみなすことができ，

$R(\vec{v},\vec{v}')=1$ iff $(\vec{v},\vec{v}')$ が有向グラフの枝

が成立する．この有向グラフは，CTL の計算モデルである Kripke 構造になっている．

3. 基本検証アルゴリズム

3.1 CTL のモデル検査

設計された順序回路 M が仕様として与えられた CTL 式 η を満たすかどうかを判定する問題を考える．

順序回路 M に対応する Kripke 構造 K (図-2) において，η の各部分式が成立する K の節点集合を求めていき，η の成立する節点集合が全ての K の初期節点を包含していれば，M は η を満たすことになる．このように，与えられた CTL 式が Kripke 構造上の全ての初期節点に対して真になるかどうかを判定することを**モデル検査 (model checking)** と言い，特に，以下に示す論理関数処理を用いたモデル検査を**記号モデル検査 (symbolic model checking)** と呼ぶ．

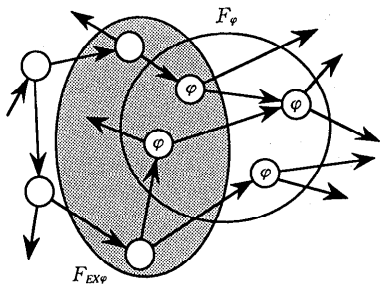


図-3　EX_φ (逆像計算)

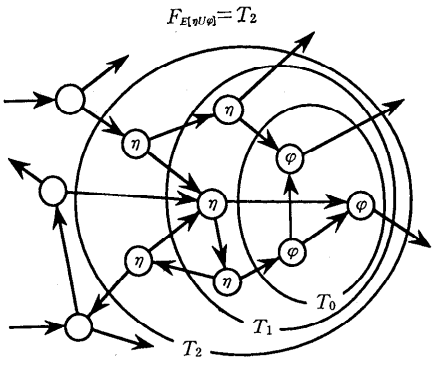


図-4　$E[\eta U\varphi]$ の計算

Kripke 構造の枝集合の特性関数を $R(\vec{v},\vec{v}')$, Kripke 構造の初期節点の集合の特性関数を $f_{init}(\vec{v})$ とする．また，η が真となるような節点集合を表現する特性関数を $F_\eta(\vec{v})$ と書く．このとき，$f_{init}(\vec{v})\Rightarrow F_\eta(\vec{v})$ が恒真になるとき，かつそのときに限り仕様 η が満たされることになる．

以下では，CTL 式 η と φ に対して，$F_\eta(\vec{v})$ と $F_\varphi(\vec{v})$ はすでに求まっているものとする．

任意のブール演算子 $\odot$ に対して，$\xi=\eta\odot\varphi$ の特性関数は $F_\xi=F_\eta\odot F_\varphi$ で与えられる．

CTL 式では，時相演算子 EX, EG, EU により他の時相演算子 AX, EF, AF, AG, AU を表すことができるので，以下では，EX, EG, EU に対する記号モデル検査アルゴリズムを示す．

$EX\varphi$ が成立する節点の集合は，**図-3** に示すように，φ が成立する節点への遷移が存在するような節点の集合となり，これは

$$F_{EX\varphi}(\vec{v})=\exists\vec{v}'.[R(\vec{v},\vec{v}')\cdot F_\varphi(\vec{v}')]$$

となる．この計算は，**逆像計算 (inverse image computation)** と呼ばれる．

$E[\eta U\varphi]$ は，**図-4** に示すように，φ が成立す

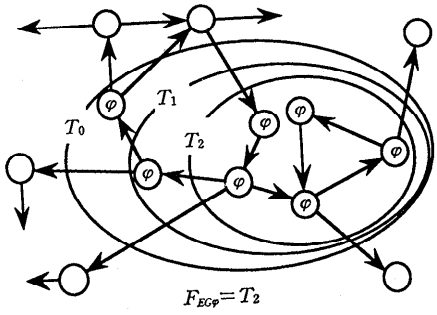


図-5 $EG\varphi$ の計算

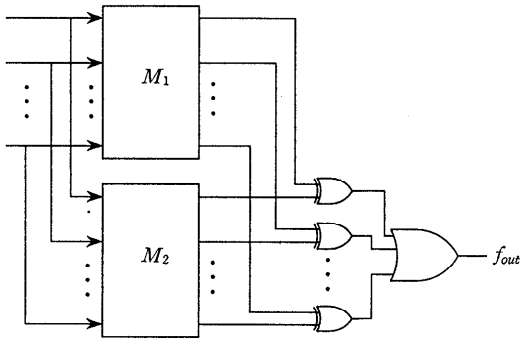


図-6 等価性判定用回路

る節点，および，η が成立する節点のみを通って φ が成立する節点へ到達可能な節点において成立する．したがって，φ の成立する節点集合に対して，η が真でその節点集合に含まれる節点への遷移が存在するような節点をその節点集合に順次追加していくことにより求めることができる．

手続き 1 ($E[\eta U\varphi]$)

procedure until ($R(\vec{v},\vec{v}')$, $F_\eta(\vec{v})$, $F_\varphi(\vec{v})$)

1. $T(\vec{v}):=F_\varphi(\vec{v})$;
2. **repeat**{
3. $U(\vec{v}):=F_\eta(\vec{v})\cdot\exists\vec{v}'.(R(\vec{v},\vec{v}')\cdot T(\vec{v}'))$;
4. **if** $U(\vec{v})\Rightarrow T(\vec{v})$ **then** $F_{E[\eta U\varphi]}(\vec{v}):=T(\vec{v})$; 終了;
5. $T(\vec{v}):=U(\vec{v})+T(\vec{v})$;
6. }

3行目では，$T(\vec{v})$ で表現される集合中のある節点へ遷移可能で，しかも，η が成立するような節点の集合を求めている．4行目の条件は，新たに求めた $U(\vec{v})$ が $T(\vec{v})$ に含まれていることを意味する．$T(\vec{v})$ が表現する集合の要素数は，図-4 の $T_0\rightarrow T_1\rightarrow T_2$ のように，単調増加することから，このアルゴリズムは必ず停止する．また，図-4 から分かるように，このアルゴリズムは逆向きの幅優先探索を行っている．

$EG\varphi$ は，**図-5** に示すように，常に φ が成立し続けるようなパスが存在する節点において成立する．これを求めるには，まず，φ が成立する節点集合から始めて，その中で次の時刻においても φ が成立し得るような節点を求め，さらに2時刻先でも φ が成立し得る節点を求めるというような操作を繰り返せば良い．

手続き 2 ($EG\varphi$)

procedure global ($R(\vec{v},\vec{v}')$, $F_\varphi(\vec{v})$)

1. $T(\vec{v}):=F_\varphi(\vec{v})$;
2. **repeat**{
3. $U(\vec{v}):=F_\varphi(\vec{v})\cdot\exists\vec{v}'.(R(\vec{v},\vec{v}')\cdot T(\vec{v}'))$;
4. **if** $U(\vec{v})=T(\vec{v})$ **then** $F_{EG\varphi}(\vec{v}):=T(\vec{v})$; 終了;
5. $T(\vec{v}):=U(\vec{v})$;
6. }

3行目は，$E[\eta U\varphi]$ の場合と同様であり，4行目で，新たに求めた $U(\vec{v})$ が $T(\vec{v})$ と等しければ $T(\vec{v})$ が解である．$T(\vec{v})$ が表現する集合の要素数は，図-5 の $T_0\rightarrow T_1\rightarrow T_2$ のように，単調減少することから，このアルゴリズムは必ず停止する．また，$E[\eta U\varphi]$ の場合と同様に，このアルゴリズムも逆向きの幅優先探索を行っている．

一般に，遷移関係関数 $R(\vec{v},\vec{v}')$ を BDD を用いて構成して逆像計算を行うと多大な記憶領域を必要とし，制約のある記憶領域上では実行不可能なことも多い．このため，種々の改良手法が提案されており，これについては，**4.** で述べる．

3.2 順序機械の等価性判定

本稿で取り扱う等価性とは，二つの順序回路 M_1 と M_2 に対して，おのおのの初期状態集合 S_1 と S_2 が与えられたとき，任意の初期状態の組 s_1 と s_2 ($s_1\in S_1, s_2\in S_2$) と任意の入力系列に対する M_1 と M_2 の出力系列が同一であることをいう．

この等価性判定問題は，**3.1** で述べた CTL の記号モデル検査により，**図-6** の回路が $AG(\neg f_{out})$ を満たすかを判定する問題に帰着することができる．一方，この問題は，図-6 の回路が $c_0=\{(s_1,s_2)|s_1\in S_1, s_2\in S_2\}$ なる初期状態集合から到達可能な全ての状態を求め，各状態における出力値が常に0であるかを調べる問題に帰着することもできる．順序回路に対して，初期状態集合から到達可能な全ての状態を求めることを**状態数え上げ**

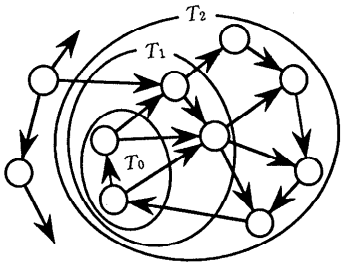


図-7 状態数え上げ

(**state enumeration**) または，**状態追跡** (**state traversal**) という．以下のアルゴリズムは，図-6のような1出力の回路の状態数え上げを行うものであり，回路に対する遷移関係関数 $R(\vec{v},\vec{v'})$，初期状態集合の特性関数 $f_{init}(\vec{v})$，および出力関数 $f_{out}(\vec{v})$ を入力として，等価性判定を行う．

手続き 3 (状態数え上げ)

procedure enumerate $(R(\vec{v},\vec{v'}), f_{init}(\vec{v}), f_{out}(\vec{v}))$

1. $T(\vec{v}) := f_{init}(\vec{v})$;
2. **repeat** {
3. **if** $T(\vec{v})\cdot f_{out}(\vec{v})\neq 0$ **then**「非等価」; 終了;
4. $U(\vec{v'}) := \exists \vec{v}.(T(\vec{v})\cdot R(\vec{v},\vec{v'}))$;
5. **if** $(U(\vec{v})\Rightarrow T(\vec{v}))=1$ **then**「等価」; 終了;
6. $T(\vec{v}) := U(\vec{v}) + T(\vec{v})$;
7. }

3行目は $T(\vec{v})$ が表現する集合中に出力を1にする状態が存在することを意味する．4行目は $T(\vec{v})$ で表現される集合に含まれる節点の子節点の集合の特性関数 $U(\vec{v})$ を求める操作を行っている．この計算は**像計算** (**image computation**) と呼ばれる．5行目の条件は，新たに求めた $U(\vec{v})$ がすでに $T(\vec{v})$ に含まれていることを意味している．$T(\vec{v})$ が表現する集合の要素数は，**図-7** の $T_0 \rightarrow T_1 \rightarrow T_2$ のように単調増加するから，このアルゴリズムは必ず停止する．また，このアルゴリズムは順方向の幅優先探索を行っている．

3.1 の逆像計算の場合と同様に，像計算は多大な計算コストを要する．その効率化手法については **4.** で述べる．

このアルゴリズムや **3.1** のアルゴリズムにおける(逆)像計算の繰返し回数は，遷移グラフの幅優先探索を行った場合の最大経路長に等しく，これが長い場合には効率が悪い．そこで，このような繰返し回数を高々 log (最大経路長) に抑えるために，グラフの推移的閉包を**反復2乗法** (**iterative squaring**) により求める方法が提案されている[8]．

まず，$Tr(\vec{v},\vec{v'})=R(\vec{v},\vec{v'})$ としてから，

$$Tr(\vec{v},\vec{v'}) := Tr(\vec{v},\vec{v'}) + \exists \vec{v''}.(Tr(\vec{v},\vec{v''})\cdot Tr(\vec{v''},\vec{v'}));$$

を繰り返す．Tr が長さ1から 2^i $(i=0,1,\cdots)$ までの全ての経路の両端の節点対の集合の特性関数になっているとすると，Tr に含まれる任意の二つの経路をつないで得られる全ての経路の両端の節点対の集合を求めて，これを元の Tr に加えることにより，長さ1から 2^{i+1} までの全ての経路の両端の節点対の集合を表す新たな Tr を得ている．この操作を Tr が変化しなくなるまで繰り返すと，グラフの推移的閉包が求まる．このとき，

$$f_{init}(\vec{v})\cdot Tr(\vec{v},\vec{v'})\cdot f_{out}(\vec{v'}) + f_{init}(\vec{v})\cdot f_{out}(\vec{v})$$

が0であれば等価，そうでなければ非等価と判定できる．

推移的閉包を求める計算では，1回の繰返し計算のコストが像計算の場合よりも大きくなるため，この手法は，従来あまり有効ではないと考えられていたが，最近になって，BDD を用いて $Tr(\vec{v},\vec{v'})$ を高速に求めるアルゴリズムが提案されている[15]．また，CTL のモデル検査に反復2乗法を用いる方法も提案されている[7]．

4. 効率化手法

4.1 像計算と逆像計算の効率化

像計算や逆像計算においては，遷移関係関数を表す BDD のサイズが大きくなり過ぎて効率低下を招くことがある．これを避けるために，遷移関係関数を分割して扱うことにより，高速化，必要記憶領域の削減を達成する手法を示す[16)~20)]．

まず，像計算の場合について考える．**2.2** で示したように，順序回路に対する遷移関係関数 $R(\vec{v},\vec{v'})$ は，$R_j(\vec{v},\vec{v'})=(y'_j \equiv f_j(\vec{v}))$ $(j=1,\cdots,m)$ とすると，

$$R(\vec{v},\vec{v'}) = \prod_{j=1,\cdots,m} R_j(\vec{v},\vec{v'})$$

となる．順序回路に対する遷移関係関数に限らず，一般にこのような $R(\vec{v},\vec{v'})$ を**積分割された遷移関係** (**conjunctive partitioned transition relation**) と呼ぶ．

たとえば，マイクロプロセッサにおいては，次

の時刻における内部レジスタの下位ビットは現時刻の内部レジスタの上位ビットにはほとんど依存しないため，マイクロプロセッサを順序回路とみなしてその状態遷移関数，すなわち，上記の $R_j(\vec{v},\vec{v}')$ を考えると，$\vec{v}$ 中の全ての変数に依存している $R_j(\vec{v},\vec{v}')$ はほとんどないと考えられる．このとき，もし $R_1(\vec{v},\vec{v}')$ は変数 y に依存するが $R_j(\vec{v},\vec{v}')$ $(j=2,\cdots,m)$ は変数 y には依存しないとすると，$\vec{v}_r$ を $\vec{v}$ から変数 y を取り除いた変数ベクトルとして，

$$\begin{aligned}&\exists\vec{v}.(T(\vec{v})\cdot R(\vec{v},\vec{v}'))\\&\quad=\exists\vec{v}_r.((\exists y.(T(\vec{v})\cdot R_1(\vec{v},\vec{v}')))\\&\qquad\cdot\prod_{j=2,\cdots,m}R_j(\vec{v},\vec{v}'))\end{aligned}$$

のように，まず $T(\vec{v})\cdot R_1(\vec{v},\vec{v}')$ に変数 y に対するスムージングを行い，その結果と $R_j(\vec{v},\vec{v}')$ $(j=2,\cdots,m)$ との論理積に残りの変数 $(\vec{v}_r)$ に対するスムージングを施すことにより像計算を行うことができる．これにより，像計算においてあらかじめ $R(\vec{v},\vec{v}')$ や $T(\vec{v})\cdot R(\vec{v},\vec{v}')$ を陽に計算しておく必要がなくなる．以下に，この考え方に基づく像計算アルゴリズムを示す．

ρ を $\{1,2,\cdots,m\}$ の置換とする．この ρ は，以下のアルゴリズムで，$R_j(\vec{v},\vec{v}')$ の処理の順番を決めるものである．ρ の決め方としては，計算途中に現れる $T_i(\vec{v},\vec{v}')$ や $T_{i-1}(\vec{v},\vec{v}')\cdot R_{\rho(i)}(\vec{v},\vec{v}')$ を表す BDD ができるだけ小さくなるように，たとえばこれらの関数が依存する変数の個数をできるだけ少なくするというようなヒューリスティクスを用いて決められる．

D_j を $R_j(\vec{v},\vec{v}')$ が依存する変数のうち $\vec{v}$ に含まれるものの集合として，変数の集合 E_i を次のように定める．

$$E_i=D_{\rho(i)}-\bigcup_{k=i+1,\cdots,m}D_{\rho(k)}$$

すなわち，E_i は $R_{\rho(i)}(\vec{v},\vec{v}')$ が依存して $\vec{v}$ に含まれる変数のうち，$R_{\rho(k)}(\vec{v},\vec{v}')$ $(k=i+1,\cdots,m)$ には現れない変数の集合であり，各 E_i は互いに素である．さらに，$\vec{v}_i$ を E_i に含まれる全ての変数による変数ベクトルとする．また，$\vec{v}$ に含まれる変数のうちどの E_i にも含まれない変数による変数ベクトルを $\vec{v}_0$ とする．このとき，$T'(\vec{v}')=\exists\vec{v}.(T(\vec{v})\cdot R(\vec{v},\vec{v}'))$ は次のように求めることができる．

$$\begin{aligned}T_0(\vec{v})&=\exists\vec{v}_0.T(\vec{v})\\T_1(\vec{v},\vec{v}')&=\exists\vec{v}_1.(T_0(\vec{v})\cdot R_{\rho(1)}(\vec{v},\vec{v}'))\\T_2(\vec{v},\vec{v}')&=\exists\vec{v}_2.(T_1(\vec{v},\vec{v}')\cdot R_{\rho(2)}(\vec{v},\vec{v}'))\\&\vdots\\T'(v')&=\exists v_m.(T_{m-1}(\vec{v},v')\cdot R_{\rho(m)}(\vec{v},v'))\end{aligned}$$

ここで述べた手法は，逆像計算にも適用することができる．その場合，上記の D_j の定義および計算手続きに出現する $\vec{v}$ と $\vec{v}'$ を入れ替えれば良い（ただし，$R(\vec{v},\vec{v}')$ の $\vec{v}'$ と $\vec{v}$ は入れ替えない）．

順序回路を対象とした逆像計算を考えた場合，$R_j(\vec{v},\vec{v}')=(y_j'\equiv f_j(\vec{v}))$ の形をしているため，$E_i=\{y'_{\rho(i)}\}$ となる．これは，逆像計算の過程で，状態遷移関数を一つずつ処理することができることを意味している．この場合，上述の計算過程は，与えられた $T(\vec{v}')$ において，各次入力変数 x_i' に対してスムージング演算を施し，各次状態変数 y_j' には対応する次状態関数 $f_j(\vec{v})$ を順次代入していくことになる．この方法をここでは**順次法 (sequential method)** と呼ぶ．この場合，ρ としては BDD の変数の順序に従って（根から葉の方向）代入操作を行うと効率が良い．また，最近になって，順序回路を対象としたより効率の良い逆像計算法として，与えられた $T(\vec{v})$ の BDD の節点を深さ優先探索しながら葉から根の方向へ上述のスムージング演算と代入演算を施していく**ボトムアップ法 (bottom-up method)** が提案されている[20),21)]．

4.2 状態集合の特性関数の簡単化

手続き1や3では，$U(\vec{v})$ と $T(\vec{v})(oldT(\vec{v})$ とする）の論理和をとって，新たに $T(\vec{v})$ として（逆）像計算を繰り返している．しかし，$cldT(\vec{v})$ から新たに出現した節点集合は $U(\vec{v})\cdot\overline{oldT(\vec{v})}$ に相当する部分だけであり，（逆）像計算で新たに $U(\vec{v})$ を求める場合，$oldT(\vec{v})$ の部分は不要であり，don't care として扱うことができる．すなわち，$(U(\vec{v})\cdot\overline{oldT(\vec{v})})\Rightarrow f'(\vec{v})\Rightarrow(U(\vec{v})+oldT(\vec{v}))$ を満足する f' であれば，どのような関数であっても（逆）像計算の $T(\vec{v})(T(\vec{v}'))$ の代わりに使うことができる．たとえば，f' が表現する集合としては，**図-8** の網掛け部分で示した集合が考えられる．

このような f' を求める演算に，restrict 演算[9),22)] や cofactor 演算[9),16),22)] がある．これらの演算は，

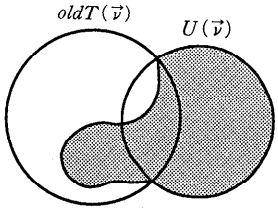


図-8 restrict/cofact 演算

論理関数 f とその care 集合の特性関数 c の BDD を引数として，$f \cdot c = f' \cdot c$ を満足する f' の BDD を返すものであり，反例はあるが，多くの場合，f' の節点数は f よりも少なくなることが知られている．

たとえば，$c = \overline{oldT(\vec{v})}$ として restrict 演算を使った状態数え上げのアルゴリズムは次のようになる．

手続き 4 (restict 演算を使った状態数え上げ)

procedure r_enumerate($R(\vec{v}, \vec{v}')$, $f_{init}(\vec{v})$, $f_{out}(\vec{v})$)

1. $f'(\vec{v}) := T(\vec{v}) := f_{init}(\vec{v})$;
2. **repeat** {
3. **if** $T(\vec{v}) \cdot f_{out}(\vec{v}) \neq 0$ **then** 「非等価」; 終了;
4. $U(\vec{v}') := \exists \vec{v}.(R(\vec{v}, \vec{v}') \cdot f'(\vec{v}))$;
5. **if** $(U(\vec{v}) \Rightarrow T(\vec{v})) = 1$ **then** 「等価」; 終了;
6. $f'(\vec{v}) = \text{restrict}(U(\vec{v}), \overline{T(\vec{v})})$;
7. $T(\vec{v}) := U(\vec{v}) + T(\vec{v})$;
8. }

この手法では，像計算や逆像計算の際に元になる節点集合を表す特性関数の BDD の節点数を削減することができるが，収束を判定するために $T(\vec{v})$ の計算が必要であり，$T(\vec{v})$ の節点数の増大が問題となる．そこで，$f_{init}(\vec{v}) \Rightarrow AG(I(\vec{v}))$ の形の CTL 式で表現される性質の検証に対しては，節点集合を表す特性関数の積分解を用いて必要記憶領域を削減する手法も提案されている[23]．

4.3 効率化手法の効果

順序回路の等価性判定を **4.1** の手法と **4.2** の restrict 演算を利用する手法を用いて行った場合，ISCAS '89 ベンチマーク回路では，両手法ともフリップフロップ数 30 個程度の二つの回路に対する等価性判定ならば，おおむね可能であることが実験的に知られている[16),22)]．

順序回路の等価性判定を行う手法として，**4.1** の手法の代わりに**再帰的像計算(recursive image computation)** を用いる手法も提案されている[16)]が，**4.1** の手法のほうが，平均的には良い結果を与えるようである[16),18)]．

CTL のモデル検査に **4.1** の遷移関係関数を積分割する手法を適用した場合，簡単なパイプラインプロセッサ（命令数は 1 ないし 2）に対する実験結果では，時間に関して 10-15 倍，必要記憶領域については 100 倍以上の改善がみられ[17)]，また，命令数を 16 に増やした場合には，**3.1** のアルゴリズムでは解くことができないが，順次法を適用することによって検証が可能になることが実験的に確認されている[19)]．

また，マイクロプロセッサの検証例では，順次法を用いる場合に比べてボトムアップ法を用いることにより時間にして 10 倍，必要記憶領域について 2 倍程度改善されることが確認されている[20),21)]．

一方，節点集合の特性関数を分割し検証を行った場合，簡単なネットワークプロトコルの検証に対して，通常の等価性判定アルゴリズムや分割を用いない手法と比べ，時間，必要領域ともに 10 倍以上改善されることが実験的に示されている[23)]．

5. あ と が き

本稿では，順序回路の設計検証に関して，二分決定グラフに基づく論理関数処理を利用する手法を，CTL のモデル検査と等価性判定について解説した．

順序回路の設計検証は，状態数爆発のため，大規模な対象に対しては，きわめて困難であるが，本稿で紹介した二分決定グラフを用いる手法は，従来の状態遷移表を用いる手法に比べ，はるかに状態数の大きな順序回路を扱うことができるという点で，実用的に重要な技法である．しかし，一方で，状態遷移関数が BDD によって少ない記憶領域で表現できたとしても，順序回路の検証がうまくいくとは限らない，という問題もある．

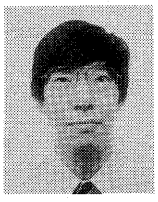
現在，大規模回路を抽象化または仕様に基づいて極小化し，その結果得られたより小さな有限状態機械を検証する手法[24),25)]など，より巨大な対象に対する検証を可能にするための研究や，実際の論理設計検証への適用[26),27)]が始まっており，ここでも論理関数処理は重要な役割を担っている．今後とも，二分決定グラフによる論理関数処理は，

順序回路や有限状態機械の検証において，有用な技術として用いられていくと考えられる．

参考文献

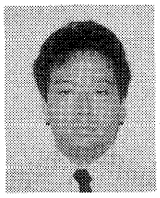
1) Rescher, N. and Urquhart, A.: *Temporal Logic*, Springer-Verlag (1971).
2) Clarke, E. M., Emerson, E. A. and Sistla, A. P.: Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach, in *10th ACM Symp. on Principles of Programming Languages*, pp. 117-126 (1983).
3) Browne, M. C., Clarke, E. M., Dill, D. L. and Mishra, B.: Automatic Verification of Sequential Ciricuits Using Temporal Logic, *IEEE Trans. Comput.*, Vol. C-35, No. 12, pp. 1035-1044 (1986).
4) Akers, S. B.: Binary Decision Diagrams, *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 509-516 (1978).
5) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677-691 (1986).
6) Bose, S. and Fisher, A.: Automatic Verification of Synchronous Circuits Using Symbolic Logic Simulation and Temporal Logic, in *Proc. IMEC-IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, pp. 759-764 (1989).
7) Burch, J. R., Clarke, E. M., McMillan, K. L. and Dill, D. L.: Sequential Circuit Verification Using Symbolic Model Checking, in *Proc. 27th Design Automation Conference*, pp. 46-51 (1990).
8) Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. and Hwang, J.: Symbolic Model Checking: 10^{20} States and Beyond, in *Proc. Logic in Computer Science* (1990).
9) Coudert, O., Berthet, C. and Madre, J. C.: Verification of Synchronous Sequential Machines Based on Symbolic Execution, in Sifakis, J. ed., *Automatic Verification Methods for Finite State Systems, LNCS 407*, Springer-Verlag (1989).
10) Coudert, O., Madre, J. C. and Berthet, C.: Verifying Temporal Properties of Sequential Machines without Building Their State Diagrams, in *Proc. Workshop on Computer-Aided Verification* (1990).
11) McMillan, K.: *Symbolic Model Checking*, Kluwer Academic Publishers (1993).
12) Bryant, R. E.: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication, *IEEE Trans. Comput.*, Vol. 4, 0 No. 2, pp. 205-213 (1991).
13) Bryant, R. E.: Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams, *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293-318 (1992).
14) 石浦：BDD とは，情報処理，Vol. 34, No. 5, pp. 585-592 (May 1993).
15) 松永，McGeer, P. C.，藤田：2分決定グラフを用いた推移的閉包計算アルゴリズムと形式的検証への応用，情報処理学会研究報告，93-DA-65, pp. 49-56 (1993).
16) Touati, H. J., Savoj, H., Lin, B., Brayton, R. K. and Sangiovanni-Vincentelli, A.: Implicit State Enumeration of Finite State Machines using BDD's, in *Proc. ICCAD*, pp. 130-133 (1990).
17) Burch, J. R., Clarke, E. M. and Long, D. E.: Representing Circuits More Efficiently in Symbolic Model Checking, in *Proc. 28th Design Automation Conference*, pp. 403-407 (1991).
18) Burch, J. R., Clarke, E. M. and Long, D. E.: Symbolic Model Checking with Partitioned Transition Relations, in *Proc. Int. Conf. on Very Large Scale Integration*, pp. 49-58 (1991).
19) Hiraishi, H., Hamaguchi, K., Ochi, H. and Yajima, S.: Vectorized Symbolic Model Checking of Computation Tree Logic, in *Proc. Workshop on Computer Aided Verification*, pp. 279-290 (1991).
20) Hiraishi, H. and Nakae, T.: An Efficient Inverse Image Computation Algorithm for Sequential Machine Verification, in *Proc. Pacific Rim Int. Symp. on Fault Tolerant Computing*, pp. 91-95 (1993).
21) Hiraishi, H., Nakae, T. and Hamaguchi, K.: Formal Verification of Single Phase Behavior of KUE-CHIP 2 Microprocessor, in *New Advances in Computer Aided Design & Computer Graphics (CAD/Grachics '93)*, Vol. II, pp. 611-616 (1993).
22) Coudert, O. and Madre, J. C.: A Unified Framework for the Formal Verification of Sequential Circuits, in *Proc. ICCAD*, pp. 126-129 (1990).
23) Hu, A. and Dill, D.: Efficient Verification with BDDs Using Implicitly Conjuncted Invariants, in *Proc. CAV '93, LNCS 697*, pp. 3-14 (1993).
24) Clarke, E. M., Grumberg, O. and Long, D. E.: Model Checking and Abstraction, in *Proc. Principles of Programming Languages* (1992).
25) Chiodo, M., Shiple, T., Sangiovanni-Vincentelli, A. and Brayton, R.: Automatic Compositiona Minimization in CTL Model Checking, in *Proc. ICCAD*, pp. 172-178 (1992).
26) 藤田，松永：形式的検証手法による実設計の検証，第6回回路とシステム軽井沢ワークショップ論文集，pp. 351-356 (1993).
27) 藤田，陳，山崎：形式的検証手法の実設計への適用例，本号，pp. 719-725 (Aug. 1994).

(平成6年1月10日受付)



平石 裕実（正会員）

1951 年生．1973 年京都大学工学部電子工学科卒業．1975 年同大学院工学研究科修士課程電気工学第二専攻修了．1975 年京都大学工学部情報工学科助手．1991 年京都産業大学工学部情報通信工学科教授，現在に至る．工学博士．形式的論理設計検証，計算機援用論理設計に興味をもつ．電子情報通信学会会員．



浜口 清治（正会員）

1964 年生．1987 年京都大学工学部情報工学科卒業．1989 年同大学院修士課程修了．1991 年京都大学工学部情報工学科助手，現在に至る．工学博士．計算機設計 CAD，特に形式的検証に関心をもつ．電子情報通信学会会員．